Relatório do Trabalho de Banco de Dados

Peterson Carvalho - GRR20163053

1 Introdução

O algoritmo de teste de serialização de escalonamento foi feito em Python 2.7, utilizando a biblioteca NetworkX para a manipulação de grafos. A biblioteca de grafos encontra-se no diretório junto com o programa principal.

2 Estrutura de Dados

2.1 Lista de Transações

A lista de transações foi guardada em uma lista de dicionários para facilitar a leitura do código. Cada dicionário representa uma linha da entrada, contendo o timestamp, id, operação e variável.

```
>>>print schedule
[{'timestamp': 1, 'id': 1, 'operation': 'R', 'variable': 'x'},
 {'timestamp': 2, 'id': 2, 'operation': 'R', 'variable': 'x'},
 {'timestamp': 3, 'id': 2, 'operation': 'W', 'variable': 'x'},
 {'timestamp': 4, 'id': 1, 'operation': 'W', 'variable': 'x'},
 {'timestamp': 5, 'id': 2, 'operation': 'C', 'variable': '-'},
 {'timestamp': 6, 'id': 1, 'operation': 'C', 'variable': '-'}]
```

2.2 Grafos

Para os grafos de precedência, foram utilizadas as funções add_node, add_edge, simple_cycles da biblioteca NetworkX e a classe DiGraph que representa um grafo orientado.

```
import networkx
#cria um grafo orientado vazio.
G = networkx.DiGraph()

#cria um nodo com identificador 'elem' (pode ser qualquer tipo)
G.add_node(elem)

#cria uma aresta do nodo x para o nodo y
G.add_edge(x, y)
```

```
#retorna uma lista de ciclos base para o grafo G
networkx.simple_cycles(G)
```

3 Algoritmo

A cada linha lida da entrada, o programa adiciona a transação à lista *schedule* e o id à lista *transactions_to_commit*. Se a operação for um *commit*, o id da transação é retirado da lista *transactions_to_commit*.

O programa só faz a verifição de serialização quando a lista *transactions_to_commit* estiver vazia. Depois da verificação, o programa printa a saída e faz tudo novamente para o próximo escalonamento que virá da entrada.

3.1 Teste de Serialização por Conflito

```
#schedule: lista de transações
#return: bool
def is_conflict_serializable(schedule):
    ...
```

Inicialmente, cria-se um nodo para cada transação do escalonamento. Depois disso, é criado uma aresta para toda leitura após escrita, escrita após leitura e escrita após escrita nas mesmas variáveis mas por transações distintas. A função retorna *False* se detecta algum ciclo em qualquer linha do escalonamento e retorna *True* se não detectar nenhum ciclo até o final do escalonamento.

3.2 Teste de Serialização por Visão

```
#schedule: lista de transações
#return: bool
def is_view_serializable(schedule):
    ...
```

Inicialmente, cria-se um nodo para cada transação do escalonamento e também os nodos T0 e Tf que escreve e lê todas as variáveis, respectivamente. Depois disso, para cada leitura de Ti em X com origem de Tj (escrita mais próxima) em X, é colocado uma aresta de Tj para Ti. Para todo Tk diferente de Ti e Tj que também escreve na mesma variável X, é colocado na lista *possible_pairs* as duas possíveis arestas (antes ou depois do par Ti, Tj).

No fim do escalonamento, é verificado se algum grafo dentre todas as combinações possíveis de *possible_pairs* não contém ciclo. Se algum não tiver ciclo, a função retorna *True* e caso contrário, retorna *False*.